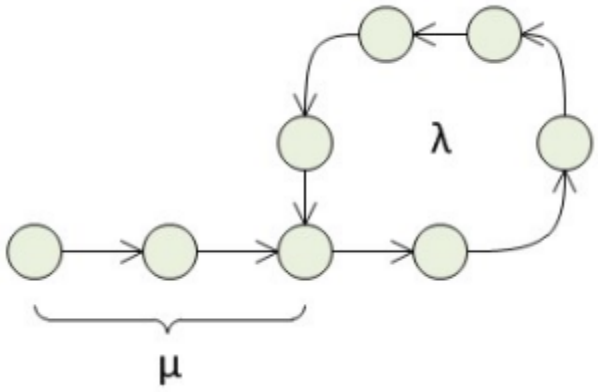
Floyd Cycle detection

Let us consider a linked list with a cycle as having a tail μ items long and a cycle λ items long.



We can always find a number m such that: mλ >= μ
mλ's position in the cycle is mλ - μ. Extra full cycles of move won't matter.
In particular: mλ + mλ and mλ will meet at mλ - μ

SO, if we have a pointer move twice each time and a pointer move once. They will meet!
The CYCLE DETECTION CODE:

```
def detect_cycle(head):
    tortoise = head
    hare = head

    while hare:
        tortoise = tortoise.next
        hare = hare.next
        if hare:
            hare = hare.next
            if tortoise is hare:
                return True
    return False
```

Once cycle detection ends, the slow stops at position (mλ- μ) in the cycle and moved mλ = μ + (mλ- μ)
Start from head with a new pointer, move 1 step each time; and the slow pointer starts moving again.
Once both pointers moved μ. low pointer's position μ + mλ treated as μ + (mλ- μ + μ)
SO on the cycle, the new pointer and the slow pointer meet, because they are at the same position in the cycle, that is the intersection node

```
tortoise = head
hare = head

# Determine if there is a cycle.
while hare:
    tortoise = tortoise.next
    hare = hare.next
    if hare:
        hare = hare.next
        if tortoise is hare:
            print(tortoise)
            break
else:
    return (False, None)

# Determine the length of the tail mu.
hare = head
mu = 0
while hare is not tortoise:
    hare = hare.next
    tortoise = tortoise.next
    mu += 1
return (True, mu)
```